Contents

1. Introduction to PKC

2. Hard problems
   - IFP
   - DLP

3. Public Key Encryptions
   - RSA
   - ElGamal

4. Digital Signatures
   - DSA, KCDSA
   - Schnorr

5. Signcryption

6. Key Exchange

7. Elliptic Curve Cryptosystems

8. Certification and PKI
1. Introduction to PKC
Key Distribution Problem of Symmetric Key Crypto

- In symmetric key cryptosystems
  - Over complete graph with \( n \) nodes, \( \binom{n}{2} = \frac{n(n-1)}{2} \) pairs secret keys are required.
  - (Example) \( n=100 \), \( 99 \times 50 = 4,950 \) keys are required
  - Problem: Managing large number of keys and keeping them in a secure manner is difficult

Secret keys are required between:
(a,b), (a,c), (a,d), (a,e), (b,c),
(b,d), (b,e), (c,d), (c,e), (d,e)
Public Key Cryptography - Concept

Using a pair of keys which have special mathematical relation. Each user needs to keep securely only his private key. All public keys of users are published.

In Encryption
Anyone can lock (using the public key)
Only the receiver can unlock (using the private key)

In Digital Signature
Only the signer can sign (using the private key)
Anyone can verify (using the public key)
## Symmetric key vs. Asymmetric Key Crypto

<table>
<thead>
<tr>
<th></th>
<th>Symmetric</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key relation</td>
<td>Enc. key = Dec. key</td>
<td>Enc. Key ≠ Dec. key</td>
</tr>
<tr>
<td>Enc. Key</td>
<td>Secret</td>
<td>Public, {private}</td>
</tr>
<tr>
<td>Dec. key</td>
<td>Secret</td>
<td>Private, {public}</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Secret Public</td>
<td>Public</td>
</tr>
<tr>
<td>Example</td>
<td>SKIPJACK AES</td>
<td>RSA</td>
</tr>
<tr>
<td>Key Distribution</td>
<td>Required (X)</td>
<td>Not required (O)</td>
</tr>
<tr>
<td>Number of keys</td>
<td>Many (X)</td>
<td>Small (O)</td>
</tr>
<tr>
<td>E/D Speed</td>
<td>Fast(O)</td>
<td>Slow(X)</td>
</tr>
</tbody>
</table>

O : merit  
X : demerit
Public Key Cryptography - Concept

- **One-way functions**
  - Given $x$, easy to compute $y=f(x)$.
  - Difficult to compute $x=f^{-1}(y)$ for given $y$.

Ex) $f(x)= 7x^{21} + 3x^3 + 13x^2 + 1 \ mod \ (2^{15}-1)$
Public Key Cryptography - Concept

- **Trapdoor one-way functions**
  - Given $x$, easy to compute $f(x)$
  - Given $y$, difficult to compute $f^{-1}(y)$ in general
  - Easy to compute $f^{-1}(y)$ for given $y$ to only who knows certain information (which we call trapdoor information)

But, easy if trapdoor info. is given.
Public Key Cryptography - Concept

- Concept
  - Overcome the problem of secret key sharing in symmetric cryptosystems
  - Two keys used: public key & private key
  - Also known as two-key cryptography or asymmetric cryptography
  - Based on (trapdoor) one-way function

\[ y = f(x) \]

- But, easy if trapdoor info. is given.
Public Key Cryptography

- **Keys**
  - A pair of (Public Key, Private Key) for each user
  - Public keys must be publicly & reliably available

- **Encryption schemes**
  - Encrypt with peer’s Public Key; Decrypt with its own Private Key
  - RSA, ElGamal

- **Digital signature schemes**
  - Sign with its own Private Key; verify with peer’s Public Key
  - RSA, DSA, KCDSA, ECDSA, EC-KCDSA ...

- **Key exchange schemes**
  - Key transport or key agreement for secret-key crypto.
  - RSA; DH(Diffie-Hellman), ECDH

- **All problems clear?**
  - New Problem: How to get the right peer’s Public Key?
  - Public key infrastructure (PKI) required
  - Certificate is used to authenticate public key
Public Key Cryptosystems

- Public key cryptography is based on hard problems.

- Encryption schemes
  - RSA: based on IFP
  - ElGamal: based on DLP

- Signature schemes
  - Signature schemes with message recovery: RSA
  - Signature with appendix: ElGamal, DSA, KCDSA

- Key exchange schemes
  - Key transport: a trusted entity TA generates and distributes key
  - Key agreement: Diffie-Hellman key agreement. Both entity take part in the key agreement process to have an agreed key
Public Key Encryption vs. Digital Signature

- **Alice’s Public Key**
- **Alice’s Private Key**
- **Bob’s private Key**
- **Bob’s public Key**

**Bob**

- Plaintext \(M\) → **E** → Ciphertext \(C\) → **D** → Plaintext \(M\)

**Alice**

- Authentic channel

**Bob**

- Plaintext \(M\) → **S** → Message + Signature \(M + s\) → **V** → Yes / No

**Authentic channel**
Public Key Cryptosystems – History

- RSA scheme (1978)
- McEliece scheme (1978)
- Rabin scheme (1979)
- Knapsack scheme (1979-): Merkle-Hellman, Chor-Rivest
- ElGamal scheme (1985)
- Elliptic Curve Cryptosystem (1985): Koblitz, Miller
- Non-Abelian group Cryptography (2000): Braid group
2. Hard Problems

IFP (Integer Factorization Problem)

DLP (Discrete Logarithm Problem)
Integer Factorization Problem (IFP)

- Problem: Given a composite number $n$, find its prime factors

Primes $p$, $q$ \[ n = pq \] 

- Application: Used to construct RSA-type public key cryptosystems

- Algorithms to solve IFP (probabilistic sub-exponential algorithms)
  - Quadratic sieve
  - General Number Field Sieve
Quadratic Sieve

- Factor $n (=pq)$ using the quadratic sieve algorithm

- Basic principle:
  Let $n$ be an integer and suppose there exist integers $x$ and $y$ with $x^2 = y^2 \pmod{n}$, but $x \neq \pm y \pmod{n}$. Then $\gcd(x-y,n)$ gives a nontrivial factor of $n$.

- Example
  Consider $n=77$
  $72=-5 \pmod{77}$, $45=-32 \pmod{77}$
  $72*45 = (-5)*(-32) \pmod{77}$
  $2^3*3^4*5 = 2^5*5 \pmod{77}$
  $9^2 = 2^2 \pmod{77}$
  $\gcd(9-2,77)=7$, $\gcd(9+2,77)=11$
  $77=11*7$  Factorization
Quadratic Sieve

Example: factor n=3837523.
(textbook p. 183)

Observe
\[ 9398^2 = 5^5 \times 19 \pmod{3837523} \]
\[ 19095^2 = 2^2 \times 5 \times 11 \times 13 \times 19 \pmod{3837523} \]
\[ 1964^2 = 3^2 \times 13^3 \pmod{3837523} \]
\[ 17078^2 = 2^6 \times 3^2 \times 11 \pmod{3837523} \]

Then we have
\[ (9398 \times 19095 \times 1964 \times 17078)^2 = (2^4 \times 3^2 \times 5^3 \times 11 \times 13^2 \times 19)^2 \]
\[ 2230387^2 = 2586705^2 \pmod{3837523} \]
\[ \gcd(2230387 - 2586705, 3837523) = 1093 \]
\[ 3837523 / 1093 = 3511 \]

\[ 3837523 = 1093 \times 3511 \quad \Leftarrow \text{succeed!} \]
Quadratic Sieve

- Quadratic Sieve algorithm: find factors of integer n
  1. Initialization: a sequence of quadratic residues $Q(x) = (m+x)^2 - n$ is generated for small values of $x$ where $m = \lfloor \sqrt{n} \rfloor$.
  2. Forming the factor base: the base consists of small primes.
     $FB = \{-1, 2, p_1, p_2, ..., p_{t-1}\}$
  3. Sieving: the quadratic residues $Q(x)$ are factored using the factor base till $t$ full factorizations of $Q(x)$ have been found.
  4. Forming and solving the matrix: Find a linear combination of $Q(x)$’s which gives the quadratic congruence. The congruence gives a nontrivial factor of $n$ with the probability $\frac{1}{2}$.

http://www.answers.com/topic/quadratic-sieve?cat=technology

- Exercise 1: Find factors of $n=4841$ using the quadratic sieve algorithm
General Number Field Sieve (GNFS)

- GNFS (general number field sieve) is the most efficient algorithm known for factoring integers larger than 100 digits.
- Asymptotic running time: sub-exponential

\[ L_n \left[ \frac{1}{3}, 1.526 \right] = O \left( e^{(1.526 + o(1))(\ln n)^{1/3} (\ln \ln n)^{2/3}} \right) \]

Complexity of algorithm

\[ L_n [\alpha, c] = O( e^{c (\ln n)^{\alpha} (\ln \ln n)^{1-\alpha}} ) \]

- If \( \alpha = 0 \), polynomial time algorithm
- If \( \alpha \geq 1 \), exponential time algorithm
- If \( 0 < \alpha < 1 \), sub-exponential time algorithm

\( \ln n \) : number of bits of \( n \)
## RSA Challenge

<table>
<thead>
<tr>
<th>Digits</th>
<th>Year</th>
<th>MIPS-year</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-100</td>
<td>'91.4.</td>
<td>7</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-110</td>
<td>'92.4.</td>
<td>75</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-120</td>
<td>'93.6.</td>
<td>830</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-129</td>
<td>'94.4.(AC94)</td>
<td>5,000</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-130</td>
<td>'96.4.(AC96)</td>
<td>?</td>
<td>NFS</td>
</tr>
<tr>
<td>RSA-140</td>
<td>'99.2 (AC99)</td>
<td>?</td>
<td>NFS</td>
</tr>
<tr>
<td>RSA-155</td>
<td>'99.8</td>
<td>8,000</td>
<td>GNFS</td>
</tr>
<tr>
<td>RSA-160</td>
<td>'03.1</td>
<td></td>
<td>Lattice Sieving + HW</td>
</tr>
<tr>
<td>RSA-174</td>
<td>'03.12</td>
<td></td>
<td>Lattice Sieving + HW</td>
</tr>
<tr>
<td>RSA-200</td>
<td>'05.5</td>
<td></td>
<td>Lattice Sieving + HW</td>
</tr>
</tbody>
</table>

- **MIPS** : 1 Million Instruction Per Second for 1 yr = $3.1 \times 10^{13}$ instruction
RSA Challenge Solution

RSA-160
Date: Tue, 1 Apr 2003 14:05:10 +0200
From: Jens Franke
Subject: RSA-160

We have factored RSA160 by gnfs. The prime factors are:
p=45427892858481394071686190649738831
   656137145778469793250959984709250004157335359
q=47388090603832016196633832303788951
   973268922921040957944741354648812028493909367

http://www.loria.fr/~zimmerma/records/rsa160

RSA-200
Date: Mon, 9 May 2005 18:05:10 +0200 (CEST)
From: Thorsten Kleinjung
Subject: rsa200

We have factored RSA200 by GNFS. The factors are
p=35324619344027701212726049781984643686711974001976
   25023649303468776121253679423200058547956528088349
and
q=79258699544783330333470858414800596877379758573642
   19960734330341455767872818152135381409304740185467

http://www.loria.fr/~zimmerma/records/rsa200
Discrete Logarithm Problem (DLP)

- **Problem:**
  Given \( g, y, \) and prime \( p \), find an integer \( x \), if any, such that
  \[ y = g^x \mod p \ (x = \log_g y) \]

- **Application:** Used to construct Diffie-Hellman & ElGamal-type public key systems: DH, DSA, KCDSA …

- **Algorithms to solve DLP:**
  - Shank’s Baby Step Giant Step
  - Index calculus
Shank’s Baby Step, Giant Step algorithm

- Problem: find an integer \( x \), if any, such that \( y = g^x \mod p \) (\( x = \log_g y \))

- Algorithm
  1. Choose an integer \( N = \left\lfloor \sqrt{p-1} \right\rfloor \)
  2. Computes \( g^j \mod p \), for \( 0 \leq j < N \)  
     - Baby Step
  3. Computes \( yg^{-Nk} \mod p \), for \( 0 \leq k < N \)  
     - Giant Step
  4. Look for a match between the two lists. If a match is found,
     \( g^j = yg^{-Nk} \mod p \)
     Then \( y = g^x = g^{j+Nk} \)
     We solve the DLP. \( x = j + Nk \)
Shank’s Baby Step, Giant Step algorithm

Match found!

\[ g^j = yg^{-Nk} \mod p \]
Problem: find an integer $x$, if any, such that $y = g^x \mod p$ ($x=\log_g y$)

Algorithm

1. Choose a factor base $S=\{p_1, p_2, \ldots, p_m\}$ which are primes less than a bound $B$.
2. Collect linear relations
   1. Select a random integer $k$ and compute $g^k \mod p$
   2. Try to write $g^k$ as a product of primes in $S$
      $$g^k = \prod_i p_i^{a_i} \mod p,$$
      then $k = \sum_i a_i \log_g p_i \mod p - 1$

3. Find the logarithms of elements in $S$ solving the linear relations

4. Find $x$
   For a random $r$, compute $yg^r \mod p$ and try to write it as a product of primes in $S$.
   $$yg^r = \prod_i p_i^{b_i} \mod p,$$
   then $x = -r + \sum_i b_i \log_g p_i \mod p - 1$
Example: Let \( p=131, g=2, y=37 \). Find \( x=\log_2 37 \) mod 131

Solution
Let \( B=10, S=\{2,3,5,7\} \)

\[
\begin{align*}
2^1 &= 2 \mod 131 \\
2^8 &= 5^3 \mod 131 \\
2^{12} &= 5 \times 7 \mod 131 \\
2^{14} &= 3^2 \mod 131 \\
2^{34} &= 3 \times 5^2 \mod 131
\end{align*}
\]

\[
\begin{align*}
1 &= \log_2 2 \mod 130 \\
8 &= 3 \times \log_2 5 \mod 130 \\
12 &= \log_2 5 + \log_2 7 \mod 130 \\
14 &= 2 \times \log_2 3 \mod 130 \\
34 &= \log_2 3 + 2 \times \log_2 5 \mod 130
\end{align*}
\]

\[
\begin{align*}
37 \times 2^{43} &= 3 \times 5 \times 7 \mod 131 \\
\log_2 37 &= -43 + \log_2 3 + \log_2 5 + \log_2 7 \mod 130 = 41
\end{align*}
\]

Solution: \( 2^{41} \mod 131 = 37 \)

Exercise 2: Let \( p=809 \). Find \( \log_3 525 \) mod 809.
Discrete Logarithm Problem (DLP)

- Complexity of best known algorithm for solving DLP:

\[ L_p \left[ \frac{1}{3}, 1.923 \right] = O \left( e^{(1.923+o(1))\left( \ln p \right)^{1/3}\left( \ln \ln p \right)^{2/3}} \right) \]

- Complexities of solving IFP and DLP are similar
3. Public Key Encryption

RSA

ElGamal
RSA Public Key Systems

- RSA is the first public key cryptosystem
- Proposed in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman at MIT
- It is believed to be secure and still widely used

[Image of Shamir, Rivest, and Adleman]

Shamir  Rivest  Adleman
RSA Public Key Systems

- **Key generation**
  - Choose two large (512 bits or more) primes p & q
  - Compute modulus n = pq, and \( \phi(n) = (p-1)(q-1) \)
  - Pick an integer e relatively prime to \( \phi(n) \), gcd(e, \( \phi(n) \))=1
  - Compute d such that ed = 1 mod \( \phi(n) \)
  - Public key (n, e) : publish
  - Private key d : keep secret (may discard p & q)

- **Special Property**
  - \((m^e \mod n)^d \mod n = (m^d \mod n)^e \mod n\) for \(0 < m < n\)

- **Encryption / Decryption**
  - E: \(c = m^e \mod n\) for \(0 < m < n\)
  - D: \(m = c^d \mod n\)
  - **Proof** \(C^d = (M^e)^d = M^{ed} = M^{k\phi(n) + 1} = M\{M^{\phi(n)}\}^k = M\)
RSA as a Trapdoor One-way Function

Message $m$ → Ciphertext $c = f(m) = m^e \mod n$

Private key
(trapdoor information)

$n = pq \ (p \ & \ q: \ primes)$

$ed = 1 \mod (p-1)(q-1)$

Public key

$n = pq \ (p \ & \ q: \ primes)$
RSA Public Key Systems

Example:

Key Generation
- \( p=3, q=11 \)
- \( n = pq = 33, \phi(n) = (p-1)(q-1) = 2 \times 10 = 20 \)
- \( e = 3 \text{ s.t. } \gcd(e, \phi(n)) = (3,20) = 1 \)
- Choose \( d \) s.t. \( ed = 1 \mod \phi(n) \), \( 3d = 1 \mod 20 \), \( d=7 \)
- Public key = \( \{e,n\} = \{3,33\} \), private key = \( \{d\} = \{7\} \)

Encryption
- \( M = 5 \)
- \( C = M^e \mod n = 5^3 \mod 33 = 26 \)

Decryption
- \( M = C^d \mod n = 26^7 \mod 33 = 5 \)
RSA Public Key Systems

Exercise 3: Provide an example of RSA key generation, encryption, and decryption for

1) p=17, q=23 (by hand calculation)
2) p=2357, q=2551 (using big number calculator)
3) p=885320963, q=238855417 (using big number calculator)

1. Key generation
2. Encryption
3. Decryption
Selecting Primes p and q for RSA

❖ How to select primes p and q?

1. $|p| \approx |q|$ to avoid ECM (Elliptic Curve Method for factoring)

2. $p-q$ must be large to avoid trial division

3. p and q are strong prime
   - $p-1$ has large prime factor $r$ (pollard’s p-1)
   - $p+1$ has large prime factor (William’s p+1)
   - $r-1$ has large prime factor (cyclic attack)
Security of RSA

- **Common Modulus attack:**
  - If multiple entities share the same modulus $n=pq$ with different pairs of $(e_i, d_i)$, it is not secure. Do not share the same modulus!

- **Cryptanalysis:** If the same message $M$ was encrypted to different users
  - User $u_1: C_1 = M^{e_1} \mod n$
  - User $u_2: C_2 = M^{e_2} \mod n$

  If $\gcd(e_1, e_2)=1$, there are $a$ and $b$ s.t. $ae_1 + be_2 = 1 \mod n$

  Then,

  $$(C_1)^a(C_2)^b \mod n = (M^{e_1})^a(M^{e_2})^b \mod n = M^{ae_1+be_2} \mod n = M \mod n$$
Security of RSA

- Cycling attack
  If \( f(f(\ldots f(M))) = f(M) \) where \( f(M) = M^e \mod n \) ?
  If a given ciphertext appears after some iterations, we can recover the plaintext at collusion point.
  Let \( C = M^e \mod n \)
  If \( (((C^e)^e)^e)^e \mod n = C^{e^k} \mod n = C \), then \( C^{e^{(k-1)}} \mod n = M \)

- Multiplicative attack (homomorphic property of RSA)
  \( (M_1^e)(M_2^e) \mod n = (M_1 \times M_2)^e \mod n \)
Attack on RSA Implementations

- **Timing attack**: (Kocher 97)
  The time it takes to compute $C^d \pmod{N}$ can expose $d$.

- **Power attack**: (Kocher 99)
  The power consumption of a smartcard while it is computing $C^d \pmod{N}$ can expose $d$.

- **Faults attack**: (BDL 97)
  A computer error during $C^d \pmod{N}$ can expose $d$. 
Security of Public Key Encryption Schemes

- Security goals
  - **One-wayness (OW):** the adversary who sees a ciphertext is not able to compute the corresponding message
  - **Indistinguishability (IND):** observing a ciphertext, the adversary learns nothing about the plaintext. Also known as semantic security.
  - **Non-malleability (NM):** observing a ciphertext for a message $m$, the adversary cannot derive another ciphertext for a meaningful plaintext $m'$ related to $m$

- Original RSA encryption is not secure
  - In IND: deterministic encryption
  - In NM: for example, from $c=m^e$, $c' = 2^ec = (2m)^e$ is easily obtained. It cannot be used in bidding scenario.
Security of Public Key Encryption Schemes

- Indistinguishability

\[ b \in \mathbb{R}[0,1] \]

Challenge: \( C = E(m_b) \)

PKE(pk, sk) \[ m_0, m_1 \]

Guess \( b \)?

The adversary wins if he guesses \( b \) correctly with a probability significantly greater than \( 1/2 \).
Security of Public Key Encryption Schemes

- Assume the existence of Decryption Oracle
  - Mimics an attacker’s access to the decryption device

- Attack models
  - **Chosen Plaintext Attack (CPA):** the adversary can encrypt any plaintext of his choice. In public key encryption this is always possible.
  - **Non-adaptive Chosen Ciphertext Attack (CCA1):** the attacker has access to the decryption oracle before he sees a ciphertext that he wishes to manipulate
  - **Adaptive Chosen Ciphertext Attack (CCA2):** the attacker has access to the decryption oracle before and after he sees a ciphertext \( c \) that he wishes to manipulate (but, he is not allowed to query the oracle about the target ciphertext \( c \).)
RSA Padding

- **RSA encryption without padding**
  - Deterministic encryption (same plaintext $\rightarrow$ same ciphertext)
  - Multiplicative property: $m_1^e \cdot m_2^e = (m_1m_2)^e \mod n$
  - Lots of attacks possible
  - Redundancy checking is essential for security

- **RSA encryption with OAEP**
  - RSA encryption after OAEP (Optimal Asymmetric Encryption Padding)
  - Proposed by Bellare and Rogaway
  - Probabilistic encoding of message before encryption
  - RSA becomes a probabilistic encryption
  - Secure against IND-CCA2
RSA with OAEP

- **OAEP → RSA encryption**
  
  \[ s = m \oplus G(r) \]
  \[ t = r \oplus H(s) \]
  \[ c = E(s, t) \]
  
  Encryption padding
  
  **RSA decryption → OAEP**
  
  \[ (s, t) = D(c) \]
  \[ r = t \oplus H(s) \]
  \[ m = s \oplus G(r) \]
  
  RSA decryption
  
  Decryption padding

OAEP looks like a kind of Feistel network.
RSA Encryption with RSA-OAEP Padding

In PKCS #1 v2.0, v2.1

- Parameter: Hash, MGF
- Input: \( M, L, (n, e) \)

\[ EM = \begin{array}{c}
00 \\
\text{masked Seed} \\
\text{MaskedDB}
\end{array} \]

\[ C = (EM)^e \mod n \]
Diffie-Hellman / ElGamal-type Systems

- **Domain parameter generation**
  - Based on the hardness of DLP
  - Generate a large (1024 bits or more) prime $p$
  - Find a generator $g$ that generates the cyclic group $\mathbb{Z}_p^*$
  - Domain parameter = \{p, g\}

- **Key generation**
  - Pick a random integer $x \in [1, p-1]$
  - Compute $y = g^x \mod p$
  - Public key (p, g, y) : publish
  - Private key x : keep secret

- **Applications**
  - Public key encryption
  - Digital signatures
  - Key agreement
ElGamal Encryption Scheme

- **Keys & parameters**
  - Domain parameter = \{p, g\}
  - Choose \(x \in [1, p-1]\) and compute \(y = g^x \operatorname{mod} p\)
  - Public key \((p, g, y)\)
  - Private key \(x\)

- **Encryption**: \(m \rightarrow (C_1, C_2)\)
  - Pick a random integer \(k \in [1, p-1]\)
  - Compute \(C_1 = g^k \operatorname{mod} p\)
  - Compute \(C_2 = m \times y^k \operatorname{mod} p\)

- **Decryption**
  - \(m = C_2 \times C_1^{-x} \operatorname{mod} p\)
  - \(C_2 \times C_1^{-x} = (m \times y^k) \times (g^k)^{-x} = m \times (g^x)^k \times (g^k)^{-x} = m \operatorname{mod} p\)
ElGamal Encryption Scheme -- Example

- **Key Generation**
  - Let $p=23$, $g=7$
  - Private key $x=9$
  - Public key $y = g^x \mod p = 7^9 \mod 23 = 15$

- **Encryption**: $m \rightarrow (C_1, C_2)$
  - Let $m=20$
  - Pick a random number $k=3$
  - Compute $C_1 = g^k \mod p = 7^3 \mod 23 = 21$
  - Compute $C_2 = m \times y^k \mod p = 20 \times 15^3 \mod 23 = 20 \times 17 \mod 23 = 18$
  - Send $(C_1, C_2) = (21, 18)$ as a ciphertext

- **Decryption**
  - $m = C_2 / C_1^x \mod p = 18 / 21^9 \mod 23 = 18 / 17 \mod 23 = 20$
4. Digital Signatures

RSA, ElGamal, DSA, KCDSA, Schnorr
Digital Signature

- Digital Signature
  - Electronic version of handwritten signature on electronic document
  - Signing using private key (only by the signer)
  - Verification using public key (by everyone)

- Hash then sign: $\text{sig}(h(m))$
  - Efficiency in computation and communication
Digital Signature

- Security requirements for digital signature
  - Unforgeability (위조 방지)
  - User authentication (사용자 인증)
  - Non-repudiation (부인 방지)
  - Unalterability (변조 방지)
  - Non-reusability (재사용 방지)

- Services provided by digital signature
  - Authentication
  - Data integrity
  - Non-Repudiation
Digital Signature

- **Digital Signature**
  - Combine Hash with Digital Signature and use PKC
  - Provide **Authentication** and **Non-Repudiation**
  - RSA; DSA, KCDSA, ECDSA, EC-KCDSA
**RSA Signature**

- **Key generation**
  - Choose two large (512 bits or more) primes \( p \) & \( q \)
  - Compute modulus \( n = pq \), and \( \phi(n) = (p-1)(q-1) \)
  - Pick an integer \( e \) relatively prime to \( \phi(n) \), \( \gcd(e, \phi(n)) = 1 \)
  - Compute \( d \) such that \( ed = 1 \mod \phi(n) \)
  - Public key \((n, e)\) : publish
  - Private key \(d\) : keep secret (may discard \( p \) & \( q \))

- **Signing / Verifying**
  - **S**: \( s = m^d \mod n \) for \( 0 < m < n \)
  - **V**: \( m =? s^e \mod n \)
  - **S**: \( s = h(m)^d \mod n \) --- hashed version
  - **V**: \( h(m) =? s^e \mod n \)

- **RSA signature without padding**
  - Deterministic signature, no randomness introduced
RSA Signature

- RSA signature forgery: Attack based on the multiplicative property of RSA.
  \[ y_1 = (m_1)^d \]
  \[ y_2 = (m_2)^d, \text{ then} \]
  \[ (y_1y_2)^e = m_1m_2 \]
  Thus \( y_1y_2 \) is a valid signature of \( m_1m_2 \)

This is an existential forgery using a known message attack.
RSA Signing with RSA-PSS Padding

- Parameter: Hash, MGF, sLen
- Input: M, (n, d)

$M' = \text{Pad} \oplus \text{Hash}(\text{salt})$

$\text{Pad} = 0x00 \ 00 \ 00 \ 00 \ 00 \ 00 \ 00 \ 00 \ (8 \text{ octets of all zeros})$

$\text{Hash} = \text{Hash}(M)$

$\text{Random octet string of sLen octets}$

$\text{Padding string: all 0x00}$

$DB = PS \ 01 \ salt$

$\text{emLen} = \lceil (|n| - 1)/8 \rceil$

$EM = \text{MaskedDB} \ H \ bc$

$S = (EM)^d \mod n$
ElGamal Signature Scheme

- **Keys & parameters**
  - Domain parameter = \{p, g\}
  - Choose \(x \in [1, p-1]\) and compute \(y = g^x \mod p\)
  - Public key (p, g, y)
  - Private key \(x\)

- **Signature generation**: (r, s)
  - Pick a random integer \(k \in [1, p-1]\)
  - Compute \(r = g^k \mod p\)
  - Compute \(s\) such that \(m = xr + ks \mod p-1\)

- **Signature verification**
  - \(y^r s \mod p =? g^m \mod p\)
    - If equal, accept the signature (valid)
    - If not equal, reject the signature (invalid)

- **No hash function…**
Digital Signature Algorithm (DSA)

Private: \( x \)
Public: \( p, q, g, y \)

\( p : 512 \sim 1024 \)-bit prime
\( q : 160 \)-bit prime, \( q \mid p-1 \)
\( g : \text{generator of order } q \)
\( x : 0 < x < q \)
\( y = g^x \mod p \)

**Signing**

Pick a random \( k \) s.t. \( 0 < k < q \)

\[
\begin{align*}
    r &= (g^k \mod p) \mod q \\
    s &= k^{-1}(\text{SHA1}(m) + xr) \mod q
\end{align*}
\]

**Verifying**

\[
\begin{align*}
    w &= s^{-1} \mod q \\
    u1 &= \text{SHA1}(m) \times w \mod q \\
    u2 &= r \times w \mod q \\
    v &= (g^{u1} \times y^{u2} \mod p) \mod q \\
    v &=? r
\end{align*}
\]
Korean Certificate-based Digital Signature Algorithm (KCDSA)

Private: $x$
Public: $p, q, g, y$

\[ z = h(Cert\_Data) \]

$p$: 768+256k (k=0 ~ 5) bit prime
$q$: 160+32k (k=0~3) bit prime, $q \mid p-1$
$g$: generator of order $q$
$x$: $0 < x < q$
$y = g^{x'} \mod p$, $x' = x^{-1} \mod q$

**Signing**

Pick a random $k$ s.t. $0 < k < q$

- $r = \text{HAS160}(g^k \mod p)$
- $e = r \oplus \text{HAS160}(z \parallel m)$
- $s = x(k - e) \mod q$

**Verifying**

\[ e = r \oplus \text{HAS160}(z \parallel m) \]
\[ m, (r, s) \]
\[ v = y^s \times g^e \mod p \]
\[ \text{HAS160}(v) =? r \]
Schnorr Signature Scheme

- **Domain parameters**
  - p = a large prime (~ size 1024 bit), q = a prime (~size 160 bit)
  - q = a large prime divisor of p-1 (q | p-1)
  - g = an element of Z_p of order q, i.e., g ≠ 1 & g^q = 1 mod p
  - Considered in a subgroup of order q in modulo p

- **Keys**
  - Private key x ∈ R [1, q-1] : a random integer
  - Public key y = g^x mod p

- **Signature generation**: (r, s)
  - Pick a random integer k ∈ R [1, q-1]
  - Compute r = h(g^k mod p, m)
  - Compute s = k – x·r mod q

- **Signature verification**
  - r =? h(y^r·g^s mod p, m)
Security of Digital Signature Schemes

- Security goals
  - **Total break**: adversary is able to find the secret for signing, so he can forge then any signature on any message.
  - **Selective forgery**: adversary is able to create valid signatures on a message chosen by someone else, with a significant probability.
  - **Existential forgery**: adversary can create a pair (message, signature), s.t. the signature of the message is valid.
Security of Digital Signature Schemes

- Attack models
  - Key-only attack: Adversary knows only the verification function (which is supposed to be public).
  - Known message attack: Adversary knows a list of messages previously signed by Alice.
  - Chosen message attack: Adversary can choose what messages wants Alice to sign, and he knows both the messages and the corresponding signatures.
5. Signcryption

Signature + Encryption
What is Signcryption?

- Provides the functions of
  - digital signature: unforgeability & non-repudiation
  - public key encryption: confidentiality

- Two birds in one stone

- Has a significantly smaller computation & communication cost compared with traditional digital envelop (signature-then-encryption)

Cost (signcryption) $<<$ Cost (signature) + Cost (encryption)
Signcryption – system setup

- Public to all
  - \( p \): a large prime
  - \( q \): a large prime factor of \( p-1 \)
  - \( g \): \( 0 < g < p \) & with order \( q \mod p \)
  - \( G \): 1-way hash
  - \( H \): 1-way hash
  - \((E,D)\): symmetric key encryption & decryption algorithms

Alice's keys:
- \( x_a \): secret key
- \( y_a \): public key
  - \( y_a = g^{x_a} \mod p \)

Bob's keys:
- \( x_b \): secret key
- \( y_b \): public key
  - \( y_b = g^{x_b} \mod p \)
**Signcryption – 1st Example**

$m \rightarrow (c,r,s)$

**Signcryption by Alice:**
1. Pick at random $x \in_R \{1, \ldots, q-1\}$
2. $w = y^x_b \mod p$
3. $k = G(w)$
4. $r = H(m, bind\_info, w)$
5. $s = x/(r + x_a) \mod q$
6. $c = E_k(m)$
7. return $(c, r, s)$

$(c,r,s) \rightarrow m$

**Unsigncryption by Bob:**
1. $w = (y_a \cdot g^r)^{s \cdot x_b} \mod p$
2. $k = G(w)$
3. $m = D_k(c)$
4. Return $m$ if $r = H(m, bind\_info, w)$
5. Return "invalid" otherwise
Signcryption – 2nd Example

\[ m \rightarrow (c, r, s) \]

Signcryption by Alice:
1. Pick at random \( x \in_R \{1, \ldots, q-1\} \)
2. \( w = y_b^x \mod p \)
3. \( k = G(w) \)
4. \( r = H(m, \text{bind\_info}, w) \)
5. \( s = x / (1 + x_a \cdot r) \mod q \)
6. \( c = E_k(m) \)
7. return \((c, r, s)\)

\[ (c, r, s) \rightarrow m \]

Unsigncryption by Bob:
1. \( w = (g \cdot y_a^r)^{s \cdot x_b} \mod p \)
2. \( k = G(w) \)
3. \( m = D_k(c) \)
4. Return \( m \) if
   \[ r = H(m, \text{bind\_info}, w) \]
5. Return "invalid" otherwise
Signcryption – 3rd Example

\[ m \rightarrow (c,r,s) \]

Signcryption by Alice:
1. Pick at random \( x \in_R \{1, \ldots, q-1\} \)
2. \( w = y_b^x \mod p \)
3. \( k = G(w) \)
4. \( r = H(m, bind\_info, w) \)
5. \( s = (x - x_a \cdot r) \mod q \)
6. \( c = E_k(m) \)
7. return \((c, r, s)\)

\[ (c,r,s) \rightarrow m \]

Unsigncryption by Bob:
1. \( w = (g^s \cdot y_a^r)^{x_b} \mod p \)
2. \( k = G(w) \)
3. \( m = D_k(c) \)
4. Return \( m \) if
   \[ r = H(m, bind\_info, w) \]
5. Return "invalid" otherwise
Major Instantiations of Signcryption

- based on DL on an Elliptic Curve
  - Zheng, CRYPTO’97
  - Zheng & Imai IPL 1998
- based on other sub-groups (e.g. XTR)
  - Lenstra & Verheul, CRYPTO2000
  - Gong & Harn, IEEE-IT 2000
  - Zheng, CRYPTO’97
- based on DL on finite field
  - Zheng, CRYPTO’97
- based on factoring / residuosity
  - Steinfeld & Zheng, ISW2000
  - Zheng, PKC2001
Signcryption vs. Signature-then-Encryption

EXP=1+1.17

(a) Signcryption based on DL

EXP=2+2

(b) Signature-then-Encryption based on RSA

EXP=3+2.17

(c) Signature-then-Encryption based on DL
DL-based Signcryption vs. Signature-then-Encryption

# of multiplications

$|p| = |n|$
DL-based Signcryption vs. Signature-then-Encryption

comm. overhead
(# of bits)

|p| = |n|

- RSA sign-enc
- Schnorr + ElGamal
- DL Signcryption

Comparison of communication overhead for different schemes with varying key sizes. The graph shows the communication overhead in bits for different key lengths, with RSA sign-encryption, Schnorr + ElGamal, and DL Signcryption compared against each other. The y-axis represents the number of bits, ranging from 0 to 25000, and the x-axis represents key sizes from 1024 to 8190 bits.
Security Proofs

- Proofs for the confidentiality and unforgeability of signcryption
  - Confidentiality --- Providing a reduction
    - from breaking the security of signcryption with respect to adaptive chosen ciphertext attacks in the flexible public key model
    - to breaking the **GAP Diffie-Hellman assumption**, in the random oracle model
  - Unforgeability --- Providing a reduction
    - from breaking the unforgeability of signcryption against adaptive chosen message attacks
    - to the **Discrete Logarithm problem**, in the random oracle model
6. Key Exchange

Diffie-Hellman
Diffie-Hellman Key Agreement Scheme

Domain Parameters

\[ p, g \]

choose \( X_a \in [1, p-1] \)

\[ Y_a = g^{X_a} \mod p \]

choose \( X_b \in [1, p-1] \)

\[ Y_b = g^{X_b} \mod p \]

\[ \text{compute the shared key} \]

\[ K_a = Y_b^{X_a} = g^{X_bX_a} \mod p \]

\[ \text{compute the shared key} \]

\[ K_b = Y_a^{X_b} = g^{X_aX_b} \mod p \]
Diffie-Hellman Problem

- **Computational Diffie-Hellman (CDH) Problem**
  
  Given $Y_a = g^{X_a} \mod p$ and $Y_b = g^{X_b} \mod p$, compute $K_{ab} = g^{X_aX_b} \mod p$

- **Decision Diffie-Hellman (DDH) Problem**

  Given $Y_a = g^{X_a} \mod p$ and $Y_b = g^{X_b} \mod p$,
  
  distinguish between $K_{ab} = g^{X_aX_b} \mod p$ and a random string

- **Discrete Logarithm Problem (DLP)**

  Given $Y = g^X \mod p$, compute $X = \log_b Y$.

The Security of the Diffie-Hellman key agreement depends on the difficulty of CDH problem.
Man in the Middle Attack in Diffie-Hellman Key Agreement

$X_b$ : private  
$Y_b = g^{X_b}$ : public

$X_a$ : private  
$Y_a = g^{X_a}$ : public

Bob computes the session key  
$K_b = Y_c^{X_b} = g^{X_c X_b}$

Alice computes the session key  
$K_a = Y_a^{X_c} = g^{X_c X_a}$

Adversary computes the both session keys  
$K_b = Y_b^{X_c} = g^{X_c X_b}$  
$K_a = Y_a^{X_c} = g^{X_c X_a}$

Problem comes from no authentication
Diffie-Hellman Key Agreement using Certified Key

Domain Parameters
\( p, g \)

- Choose \( X_a \in [1, p-1] \)
  \( Y_a = g^{X_a} \mod p \)

Certified key
\( Y_a \) and \( Y_b \)

- Choose \( X_b \in [1, p-1] \)
  \( Y_b = g^{X_b} \mod p \)

- Compute the shared key
  \[ K_a = Y_b^{X_a} = g^{X_bX_a} \mod p \]

- Compute the shared key
  \[ K_b = Y_a^{X_b} = g^{X_aX_b} \mod p \]

- Interaction is not required
- Agreed key is fixed, long-term use
MTI Protocols -- by Matsumoto, Takashima, Imai

**Domain Parameters**

- \( p, g \)

**Certified key**

- \( Y_a \) and \( Y_b \)

**Choose**

- \( X_a \in [1, p-1] \)
- \( Y_a = g^{X_a} \text{ mod } p \)

- **Choose**\( k_a \in [1, p-1] \)
- \( T_a = g^{k_a} \text{ mod } p \)

**Compute the shared key**

- \( K_a = Y_a^{k_a} T_a^{X_a} = g^{X_b k_a} g^{k_b X_a} \)

**Choose**

- \( X_b \in [1, p-1] \)
- \( Y_b = g^{X_b} \text{ mod } p \)

- **Choose**\( k_b \in [1, p-1] \)
- \( T_b = g^{k_b} \text{ mod } p \)

**Compute the shared key**

- \( K_b = Y_a^{k_b} T_a^{X_b} = g^{X_a k_b} g^{k_a X_b} \)
7. Elliptic Curve Cryptosystem
Elliptic Curve (1)

- **Weierstrass form of Elliptic Curve**
  \[ y^2 + a_1 xy + a_3 = x^3 + a_2 x^2 + a_4 x + a_6 \]

- **Example (over rational field)**
  \[ y^2 = x^3 - 4x + 1 \]
  \[ E(Q) = \{(x,y) \in \mathbb{Q}^2 | y^2 = x^3 - 2x + 2\} \cup \mathbb{O}_E \]
  \[ P = (2, 1), \quad -P = (2, -1) \]
  \[ [2]P = (12, -41) \]
  \[ [3]P = (91/25, 736/125) \]
  \[ [4]P = (5452/1681, -324319/68921) \]
Elliptic Curve (2)

- **Example (over finite field GF(p) : p = 13)**
  - $P = (2,1)$, $\#P = (2, 12)$, $[2]P = (12, 11)$
  - Hasse’s Theorem: $p - 2\sqrt{p} \leq \#E(p) \leq p + 2\sqrt{p}$
  - Scalar multiplication: $[d]P$

- **Elliptic Curve Discrete Logarithm**
  - Base of Elliptic Curve Cryptosystem (ECC)

\[
y = g^x \mod p \quad \iff \quad Q = [d]P
\]

Find $x$ for given $Y$  
Find $d$ for given $Q$
Elliptic Curve Cryptosystems

- **Advantages**
  - Breaking PKC over Elliptic Curve is much harder
  - We can use much shorter key
  - Encryption/Decryption is much faster than that of other PKCs
  - It is suitable for restricted environments like mobile phone, smart card

- **Disadvantages**
  - It’s new technique ➔ There may be new attacks
  - Too complex to understand
  - ECC is a minefield of patents
    : e.g. US patents
    4587627/739220 – Normal Basis, 5272755 – Curve over GF(p)
    5463690/5271051/5159632 – p=2^q-c for small c, etc…
Key Sizes and Algorithms

- **System strength, Symmetric Key strength, Public Key strength**
  must be consistently matched for any network protocol usage.

- **Selection Rules**
  - Determine symmetric key sizes: \( n \)
  - Symmetric Cipher \( \rightarrow \) Key exchange Algorithm \( \rightarrow \) Authentication Algorithm

<table>
<thead>
<tr>
<th>Sym.</th>
<th>RSA/DH</th>
<th>ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td>90</td>
<td>1024</td>
<td>160</td>
</tr>
<tr>
<td>120</td>
<td>2048</td>
<td>210</td>
</tr>
<tr>
<td>128</td>
<td>2304</td>
<td>256</td>
</tr>
</tbody>
</table>

From Peter Gutmann's tutorial

<table>
<thead>
<tr>
<th>Sym.</th>
<th>RSA/DH</th>
<th>ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>430</td>
<td>112</td>
</tr>
<tr>
<td>80</td>
<td>760</td>
<td>160</td>
</tr>
<tr>
<td>96</td>
<td>1020</td>
<td>192</td>
</tr>
<tr>
<td>128</td>
<td>1620</td>
<td>256</td>
</tr>
</tbody>
</table>

From RSA's Bulletin (2000. 4. No 13)

- **Recommendation for RSA/ECC**
  - 512/112-bit: only for micropayment/smart card
  - 1024/160-bit: for short term (1-year) security
  - 2048/256-bit: for long term security (CA, RA)
Implementation Results

- **RSA Encryption/Decryption**

<table>
<thead>
<tr>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#1-v1.5</td>
<td>1.49 ms</td>
</tr>
<tr>
<td>PKCS#1-OAEP</td>
<td>1.41 ms</td>
</tr>
</tbody>
</table>

- **Signature**

<table>
<thead>
<tr>
<th>Signing</th>
<th>Verifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#1-v1.5</td>
<td>18.07 ms</td>
</tr>
<tr>
<td>PKCS#1-PSS</td>
<td>18.24 ms</td>
</tr>
<tr>
<td>DSA with SHA1</td>
<td>2.75 ms</td>
</tr>
<tr>
<td>KCDSA with HAS160</td>
<td>2.42 ms</td>
</tr>
</tbody>
</table>

- **Modular Exponentiation vs. Scalar Multiplication of EC**

<table>
<thead>
<tr>
<th>M.E. (1024-bit)</th>
<th>S.M. (GF(2^{162}))</th>
<th>S.M. (GF(p))</th>
</tr>
</thead>
<tbody>
<tr>
<td>52.01 ms</td>
<td>2.24 ms</td>
<td>1.17 ms</td>
</tr>
</tbody>
</table>
Implementation Environments

- **RSA Encryption/Signature**
  - N : 1024 bits, public exponent : $65537 = 2^{16} + 1$
  - Decryption/Signing uses Chinese Remainder Theorem (CRT)
    - CRT is roughly 3 times faster

- **DSA/KCDSA**
  - p : 1024-bit prime, q : 160-bit subprime
  - Signing uses LL-method
  - Verifying uses double-exponentiation

- **Modular Exponentiation vs. Scalar Multiplication of EC**
  - M.E./S.M. uses Window-method
  - In the same security level, ECC is much faster than RSA/DSA
8. Certification and PKI
Key Distribution Center (KDC)

- Rely on the absolute security of KDC
- Ease of centralized management
- Suitable for enterprise network security
- But not Scalable; KDC is a potential Bottleneck
Diffie-Hellman Key Exchange and Message Encryption

Bob’s Public Key/Private Key

Alice’s Public Key/Private Key

Diffie-Hellman Algorithm

Shared Secret Key

Identical!!

Encrypted Message

Symmetric Key Cryptosystem

Encryption

Decryption

86
Digital Enveloping: Key Transport + Encryption

Bob's Public Key/Private Key
Alice's Public Key
Randomly Generated Session Key

1. Hash Algorithm
2. Hash
3. Signing
4. Signature
5. Encrypted Session Key
6. Encryption
7. Encrypted Message
8. Public Key Cryptosystem
9. SEND
Digital Enveloping: Key Recovery + Decryption

- Alice’s Public Key/Private Key
- Bob’s Public Key

Encrypted Session Key

Decryption

Encrypted Message

Decryption

Signature

Hash Algorithm

Hash1

Hash2

Signature

Encrypted Session Key

Encrypted Message

RECEIVE
How to Guarantee Authenticity of Peer Public Key?

- For secure use of public key systems,
  - Everyone should be able to obtain the public key of any communication peer that he wants to communicate with, in such a way that he can be sure that the obtained public key is the correct and right public key of the peer

  - How to guarantee that the public key obtained is the right one?

  - How to guarantee that the public key obtained is authentic?

- Using Certificate!
What is a Digital Certificate?

- **Digital Certificate**
  - A file containing **Identification information** (CA’s name (Issuer), Alice’s name (Subject), valid period, Alice’s public key, etc) and **digital signature** signed by trusted third (CA) to guarantee its authenticity & integrity.

- **Certificate Authority (CA)**
  - Trusted third party like a government for passports
  - CA authenticates that the public key belongs to Alice
  - CA creates Alice’s a Digital Certificate
Data encrypted using secret key exchanged using some public key associated with some certificate.
X.509 V3 Certificate Format

Certificate ::= SEQUENCE {
    tbsCertificate   TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue   BIT STRING }

TBSCertificate ::= SEQUENCE {
    version                    [0]  EXPLICIT Version DEFAULT v1,
    serialNumber             CertificateSerialNumber,
    signature                 AlgorithmIdentifier,
    issuer                    Name,
    validity                  Validity,
    subject                   Name,
    subjectPublicKeyInfo      SubjectPublicKeyInfo,
    issuerUniqueID           [1]  IMPLICIT UniqueIdentifier OPTIONAL,
    -- If present, version shall be v2 or v3
    subjectUniqueID           [2]  IMPLICIT UniqueIdentifier OPTIONAL,
    -- If present, version shall be v2 or v3
    extensions                [3]  EXPLICIT Extensions OPTIONAL,
    -- If present, version shall be v3
}
Certificate:

Data:

Version: v3 (0x2)
Serial Number: 3 (0x3)

Signature Algorithm: PKCS #1 MD5 With RSA Encryption
Issuer: OU=Ace Certificate Authority, O=Ace Industry, C=US

Validity:
Not After: Sun Oct 17 18:36:25 1999

Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US

Subject Public Key Info:

Algorithm: PKCS #1 RSA Encryption

Public Key:

Modulus:
fb:2e:8f:fb:

Public Exponent: 65537 (0x10001)

Extensions:

Identifier: Certificate Type
Critical: no

Certified Usage:
SSL Client

Identifier: Authority Key Identifier
Critical: no

Key Identifier:
26:c9

Signature:

Algorithm: PKCS #1 MD5 With RSA Encryption
Signature:
fb:2e:8f:fb:

-----BEGIN CERTIFICATE-----
MIICzkCCAZSgAwIBAgIBAzANBgkqhkiG9w0BAQQQFADA3MQswCQYDVQQGEwJF
aXJjZUSHMRUQDAWcCQYDVQQKEVRvY3VtZW50YXJuLTQwMjAkBgNV
HSMEGDAWgBhRUxwEQQzCzAJBgNVBAYTAlVTMDcGCSqGSIb3DQEBCwUAA4
MAwGCSqGSIb3DQEAwIBBQICAQIBaGFuYW1vbmUgd2l0aCBsYXN0ZSB3
ZXNzaW5lc3QgaXMgc2VjdjguY29tMBMGA1UEAHRlMREwDwYDVQQK
MBMGA1UEAHRlMREwDwYDVQQK
MAwGCSqGSIb3DQEBAQAAAAATANBgkqhkiG9w0BAQQFAAOBgQ
-----END CERTIFICATE-----
This certificate is intended to:
  • Guarantee the identity of a remote computer

Issued to: wwws.ameritrade.com
Issued by: Secure Server Certification Authority
Valid from 6/8/00 to 6/9/01

Issued to: Secure Server Certification Authority
Valid from 11/8/94 to 1/7/10
How to Revoke a Certificate?

- Certificate Revocation List (CRL)
  - A digital document which has a list of revoked certificates
  - Signed by CA
  - Defined in X.509 v2

- Why revoke a certificate?
  - When the user leave (retire from) the organization
  - Lost the private key, need to use a new key
Certificate Revocation List

Certificate Revocation List Information

- Field: Value
  - Version: V1
  - Issuer: VeriSign Commercial Software Publisher...
  - Effective date: Monday, October 01, 2001 5:00:07 AM
  - Next update: Thursday, October 11, 2001 5:00:07 AM
  - Signature algorithm: md5RSA

Revoked certificates:

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Revocation date</th>
</tr>
</thead>
<tbody>
<tr>
<td>7019 AAC3 5401 3292 ...</td>
<td>Wednesday, May 03, 2000 5:19:20 PM</td>
</tr>
<tr>
<td>7038 003F F284 A0A8 ...</td>
<td>Tuesday, March 27, 2001 9:50:49 AM</td>
</tr>
<tr>
<td>703A 71E7 5193 3062 ...</td>
<td>Wednesday, April 11, 2001 3:34:05 AM</td>
</tr>
<tr>
<td>704B D594 A408 4BD0 ...</td>
<td>Monday, May 21, 2001 2:57:32 PM</td>
</tr>
<tr>
<td>704F D63C B010 9E95 ...</td>
<td>Tuesday, April 03, 2001 12:54:44 PM</td>
</tr>
<tr>
<td>705D D362 84A6 A324 ...</td>
<td>Wednesday, December 06, 2000 8:4</td>
</tr>
</tbody>
</table>

Revocation entry

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial number</td>
<td>7038 003F F284 A0A8 830F 4EFA 4...</td>
</tr>
<tr>
<td>Revocation date</td>
<td>Tuesday, March 27, 2001 9:50:49 AM</td>
</tr>
</tbody>
</table>
X.509 V2 Certificate Revocation List (CRL) Format

CertificateList ::= SEQUENCE {
    tbsCertList             TBSCertList,
    signatureAlgorithm     AlgorithmIdentifier,
    signatureValue         BIT STRING  }

TBSCertList ::= SEQUENCE {
    version                 Version OPTIONAL,
                           -- if present, shall be v2
    signature               AlgorithmIdentifier,
    issuer                  Name,
    thisUpdate              Time,
    nextUpdate              Time OPTIONAL,
    revokedCertificates     SEQUENCE OF SEQUENCE  {
                           userCertificate           CertificateSerialNumber,
                           revocationDate            Time,
                           crlEntryExtensions        Extensions OPTIONAL
                           -- if present, shall be v2
                       } OPTIONAL,
    crlExtensions           [0]  EXPLICIT Extensions OPTIONAL
                           -- if present, shall be v2
}
Overall Configuration of CA System

- Internet/Intranet/Extranet
- FW/IDS
- Switching HUB
- DAMS
- Web System
- Directory System
- State Information System
- Administrator System
- Client System
- DBMS
Public Key Infrastructure (PKI) Architecture

PKI is the hardware, software, people, policies, & procedures needed to create, manage, store, distribute, & revoke certificates.
PKI Trust Relationship

Hierarchical Structure

Network Structure
How a PKI works?

1. **Generate Registration Info & Keypair**
   - RA delivers the Certificate to user

2. **Send the Public Key and Registration Info to RA**
   - CA signs a valid request
   - Cert is published in Directory
   - Send the signed request back to RA

3. **Certificate Request**
   - RA delivers the Certificate to user

Applications using Certificates can:
- Look up certificate details
- Perform revocation checks
- Check certificate validity
- Check signatures
- Decrypt data

Directory

Applications and other users
Certification Hierarchy

Sub CA certificate is signed by its superior CA

Root CA
Issuer = Root
Subject = Root

Subordinate CA
Issuer = Root
Subject = Taejon

Subordinate CA
Issuer = Taejon
Subject = KAIST

End Entity
Issuer = KAIST
Subject = KAIST

End Entity
Issuer = Taejon
Subject = Taejon Citizen

End Entity
Issuer = Future
Subject = Future Employee

EE certificates are signed by their own CA

If you trust the CA that signed the certificate, you can trust the certificate
Korean PKI Structure

전자서명 인증관리센터
http://www.kisa.or.kr/kisa/kcac/jsp/kcac.jsp
Korean PKI Structure

전자서명법 제4조의 규정에 의하여 지정된 공인인증기관
  • 한국정보사회진흥원 http://www.signgate.com
  • (주)코스콤 http://www.signkorea.com
  • 금융결제원 http://www.yessign.or.kr
  • 한국정보사회진흥원 http://sign.nca.or.kr
  • 한국전자인증(주) http://gca.crosscert.com
  • 한국무역정보통신 http://www.tradesign.net
Homework #6

☐ Solve the exercises in this lecture

Exercise 1: factorization using the quadratic sieve algorithm
Exercise 2: Solve DLP using index calculus
Exercise 3: RSA construction